Towards Sketching Interfaces for Multi-Paradigm Modeling

Simon Van Mierlo, Julien Deantoni, Loli Burgueño, Clark Verbrugge, Hans Vangheluwe

Towards Sketching Interfaces for Multi-Paradigm Modeling

Simon Van Mierlo
University of Antwerp
Flanders Make vzw
simon.vanmierlo@uantwerpen.be

Julien Deantoni
Université Cote d'Azur - Sophia Antipolis
julien.deantoni@univ-cotedazur.fr

Loli Burgueño
IN3, Open University of Catalonia
Institut LIST, CEA, Université Paris-Saclay
lburguenoc@uoc.edu

Clark Verbrugge
McGill University
clump@cs.mcgill.ca

Hans Vangheluwe
University of Antwerp
Flanders Make vzw
hans.vangheluwe@uantwerpen.be

***Abstract*—Existing design processes typically begin with informal ideation by sketching out a basic approach that can be further developed into a more complete design. Although intuitively simple, and seemingly informal, the sketching process is actually a structured activity that strongly influences the design of the system; hence, it has an important role in the design success. In this work, we develop a well defined specification of the sketching activity. We consider *sketching* as a process of achieving agreement, based on stakeholders communicating ideas about a design and its properties, with the side-effect of incrementally developing a (set of) common language(s) specific to the idea domain. Our perspective on sketching further differs from more common notions of ideation by noting the roles of requirements and system properties, and offering a general perspective on sketching as a modular activity within design. We validate our approach by analyzing the sketches of a research group at the CAMPaM 2019 workshop. By recognizing sketching as a fundamental activity in design, we enhance the formalization of the design process, and suggest improvements to the tool support for sketching beyond the basic drawing features.**

***Index Terms*—sketching, multi-paradigm, ideation, interface**

I. Introduction

Imprecise drawings or *sketches* are commonly used during the design process in many domains. Core to the early *ideation phase* of the project—although also employed throughout the design process—sketches are produced on a transient medium, using informal imagery, text, and "back-of-the-envelope" calculations to explore solutions and to develop candidate designs. In collaborative settings, such drawings can be used to reach a common understanding of a problem and to express new ideas that are either discarded or refined in detailed designs.

Informal sketching is recognized as an important part of the design process [15], but also sits uneasily with more formal approaches to design and modeling. In Multi-Paradigm Modeling (MPM) [11], for example, the current assumption is that a (domain-specific) language is designed first, and then used by engineers to specify the system. When engineers are sketching, however, they do not necessarily use a well-defined modeling language: their vocabulary is incomplete and changes rapidly, with the freeform properties of sketching used to facilitate understanding and exploration.

Sketching has particular significance when applying MPM to design Cyber-Physical Systems (CPSs) [14]. In this context, sketching is an essential activity for cross-discipline brainstorming to reach a common understanding, and to quickly evaluate possible design candidates. Despite its importance, the sketching activity is currently performed in an ad-hoc way. While significant effort has been devoted to building sketching tools, little effort has been spent to identify the languages, elements, and processes used while sketching [2]. In this work we argue that, even if aimed at freeform expression, sketching has definable structure, and a more precise definition of sketching is possible that can expose the processes and artefacts involved. Better formalization of sketching enables tighter integration into formal design, allowing sketching artefacts to more seamlessly participate in traceability and other important parts of the complex CPS design process.

We validate our definitions by looking at the sketching activities performed by a research group of CAMPaM 2019[1], the workshop where we developed these ideas. We describe our envisioned approach, which revolves around model and language co-design: while a (team of) user(s) is sketching, a vocabulary and its semantics is gradually built up. Technical solutions to support these activities are proposed for dealing with imprecise syntax, uncertainty in the semantics, and supporting implementation platforms.

II. Background

This section provides background for the rest of the paper: we discuss ideation and sketching in different (engineering) disciplines, traditional system development workflows, and language engineering techniques, to provide an overview of the state-of-the-practice in sketching tools and to contrast this to typical system development workflows and tools.

This research was partially supported by Flanders Make vzw, the strategic research center for the manufacturing industry and Spanish research project TIN2016-75944-R.

[1] http://msdl.cs.mcgill.ca/conferences/CAMPaM/2019/

A. Ideation and Sketching

In (engineering) disciplines, ideation is a process used to come up with innovative ideas [18]. In design, it is commonly synonymous with "brainstorming": generating a large set of diverging ideas, and then consolidating by evaluating design candidates. Many techniques exist to improve the effectiveness of this activity, such as starting from existing solutions and transferring ideas from different fields. The ideation phase results in a set of design candidates that, with a certain degree of certainty, have been found to be suitable. These candidates can then be used as input for the detailed design process, where they are refined, tested, and ultimately deployed.

Sketching is an activity often used during ideation to convey ideas. Such sketches are mostly diagrammatic in nature, and allow users to write down their ideas for different purposes: 1) to understand the problem better; 2) to reach a common understanding; and 3) to form agreement on the problem at hand, and how to tackle it. In [16], the process of collaborative sketching in engineering is detailed. The authors note that sketches are misinterpreted frequently by participants due to their informal nature. However, they note that in certain domains such as the electrical, fluid and thermal domains, more rigor is introduced because a symbolic format is used. In the rest of the paper, we assume that the sketching activity is performed collaboratively and that the language used for sketching is important for its effectiveness.

With regards to tooling, Stappers and Hennesey [17] explore the idea of "electronic napkins" for visual ideation and arrive at the conclusion that traditional tools are too rigid, hence engineers prefer not to use them and continue using pen and paper. In contrast, Jonson [6] concluded from his study that computer tools unexpectedly had a large role in the ideation process. Based on these studies, it seems reasonable to assume that designers would use digital tools if a proper language and process support are offered.

B. System Development Workflows

The goal of a system development workflow is to come up with a system design (in a timely manner) that satisfies the properties that were formalized in the requirements engineering phase. Properties state facts about the structure of the system (a car needs to have four wheels) or its performance (the car's fuel efficiency needs to be high). Many workflows have been developed [13]: from the rigid waterfall model, to more recent agile approaches where a running prototype is always available. In system development, the V-model is a popular workflow: design and tests are developed in tandem.

Workflows are most often high-level and assume the requirements are fixed; only in agile approaches, changing requirements are inherently taken into account. However, the *early-stage* activities are often not considered in these workflows. In the ideation phase, often multiple ways of solving the problem are explored, and the requirements are subject to change. Moreover, the language used to express the early designs is most often informal, making the (tool-supported) evaluation of these early designs more difficult.

C. Language Engineering

Engineers use modelling languages to specify their system designs; these specifications can be simulated, tested, and used as a blueprint for the final system implementation. Many modelling languages exist, either tailored to a specific domain or generic within a certain paradigm. Generally, domain experts work with a small number of languages that are specialized to their specific domain.

A modelling language, be it general-purpose or domain-specific, has *syntax* and *semantics*. Most commonly, a language engineer builds the syntax and semantics of a language and the tooling (such as model editors, simulator, code generators, and debuggers) around it; these tools are then used by domain experts to design their systems. Typically, a precise meaning is assumed for the language that allows to obtain a result (e.g. the fuel efficiency of a car is x) or an execution trace that can be analyzed for temporal properties such as "when a command for shifting gears is given by the driver, the system responds within a time delay of at most y time units".

III. Definitions

As a first step to better support the sketching activity in the design process, we identify the concepts and relations involved in a sketching activity in Figure 1. The focus is on the link between various languages shared among stakeholders and the link with the design activity and ideation; we leave the study of the relations between the different languages for future work. As far as the authors are aware, this is the first attempt to formalize the sketching process from a language and process perspective; its relevance is evaluated in the next section.

A sketching activity involves a set of *Stakeholder*s, who know some *Language*s. At the start of sketching, a set of *Input*s refer to a potentially empty set of *Requirement*s, a potentially non-existing *PartialDesign* and a set of *Problem-Statement*s that may refer to the partial design. The problem statement is proposed by at least one of the stakeholders.

The sketching activity usually starts with an explanation of the problem and relies on the illustration through examples expressed in a specific language. This language is usually incomplete—both in its syntax (concepts that the users want to express in detailed design are missing) and semantics (the meaning of language elements is not yet known, defined imprecisely, or includes uncertainty). Furthermore, if the language in which the example is explained is not precisely known by all the stakeholders, it must be explained so that they understand the problem properly. This explanation about the syntax and/or semantics of the language must be carried our either by means of oral explanations, annotations of the models or expressed in a shared language.

Once the problem is understood, the stakeholders start proposing potential *Decision*s on the solution to the problem—once again, in a language that may require explanations. These decisions are evaluated, usually intellectually by the stakeholders according to the requirements and the problem statement. Once the sketching activity is considered successful,

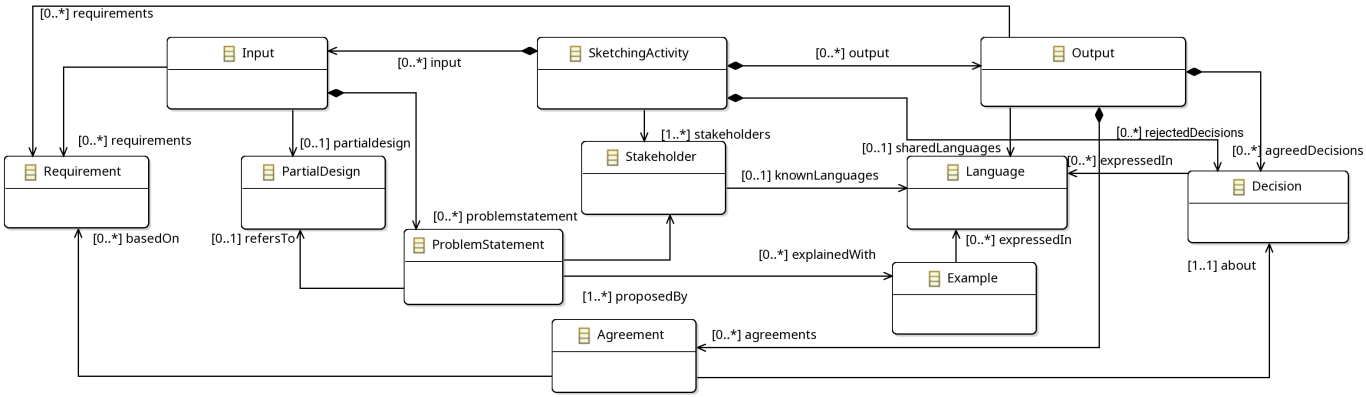


Fig. 1: Concepts and relations involved in a sketching activity.

the stakeholders reach an *Agreement* on a specific (set of) decision(s), based on the requirements. The *Output* of a sketching activity is a set of languages shared by the stakeholders (that may be reused in future sketching activities) and a potentially empty set of agreements on decisions, expressed in at least one of the shared language(s). The nature of the relations between the different models and languages used for the requirements, the partial design and the decisions taken during a sketching activity are topics to be investigated. However, we focus in this paper on the models and languages used, created and/or modified during the sketching (sub-)activities.

The workflow in which the artefacts used during the sketching subactivities is described in Figure 2 by a Formalism Transformation Graph + Process Model [8] (FTG+PM) model. The right part shows the different activities described earlier and the input/output of each activity. The left part shows a "map" of all the formalisms used during the process: each artefact created in the workflow is typed by a language in the formalism transformation graph. Similarly, each (manual) activity in the workflow is typed by a (manual) transformation between two languages.

The activity of understanding the problem is iterative and there may be four (or more) languages involved: 1) the language to describe the problem (natural language); 2) the language to explain the problem through examples (*Problem Explanation Language (PEL)*); 3) the language to specify the requirements (*Req. Lan.*); and 4) the meta formalism to specify the requirement language and the example language (*MetaFormalism*). An instance of these three languages (an example in case of the PEL, a requirement in case of the Req. Lan., and a language in case of MetaFormalism) are both inputs and outputs of the *UnderstandProblem* activity, since they are usually modified during the activity. Also, while not on Figure 2 for readability, several different languages can be used to describe one or different examples. Once the problem is understood, the *DescribeDecision* activity starts: some decisions are elaborated step by step by the stakeholders, with the support of one or more *Domain Specific Language(s) (DSLs)*. These DSLs often also evolve in the process of decision making. Both the decision and the language used to specify them are therefore inputs and outputs of the activity. Finally, a decision is evaluated according to the requirements and can either be accepted or rejected. In the former case, the decision is the source of an agreement and the sketching activity ends. In the latter case, the decisions have to be evolved until agreement is reached.

IV. Case Study

To verify the relevance of the identified concepts and relations, we reviewed the sketches produced by a group at the CAMPAM 2019 workshop. These artefacts were realized on a blackboard by three stakeholders (researchers in computer science) working on the generalization of adaptive abstraction for multi-agent systems. For traceability reasons, the stakeholders took pictures of the board at different points in time; these are shown in Figures 3a, 3b, 3c, ordered chronologically from left to right. We kindly asked our subjects to explain their sketches. Figure 3a contains examples used by one of the stakeholders to explain the problem. We highlighted in red the concrete syntax used to express the example and in green the concrete syntax used to explain the semantics of the diagram. In Figure 3b, the stakeholders started proposing a decision to solve the problem (highlighted in blue). Finally, Figure 3c contains a refinement of a decision the stakeholders reached to solve the initial problem. Of course, throughout the sketching process, they used oral explanations, too.

Clearly, in this case study we find the concepts that we identified in the previous sections, hence it validates our proposed definitions. Furthermore, the stakeholders decided to keep track of their thinking (by taking pictures of the board). However, there is room for improvement. First, they left the initial requirements implicit. Second, they did not identify/classify the various artefacts on the board (it may be interesting to ask them 6 months later what their understanding of such artefacts is). Last, they did not keep track of their rejected decisions, which could prevent other people reusing the artefacts to go in the same (potentially wrong) direction.

V. Tool Support Roadmap

This section describes how we envision the tool support for sketching should evolve in the future to support the

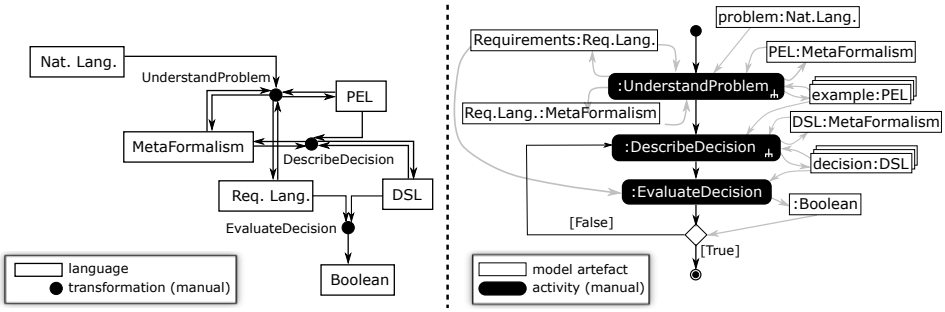
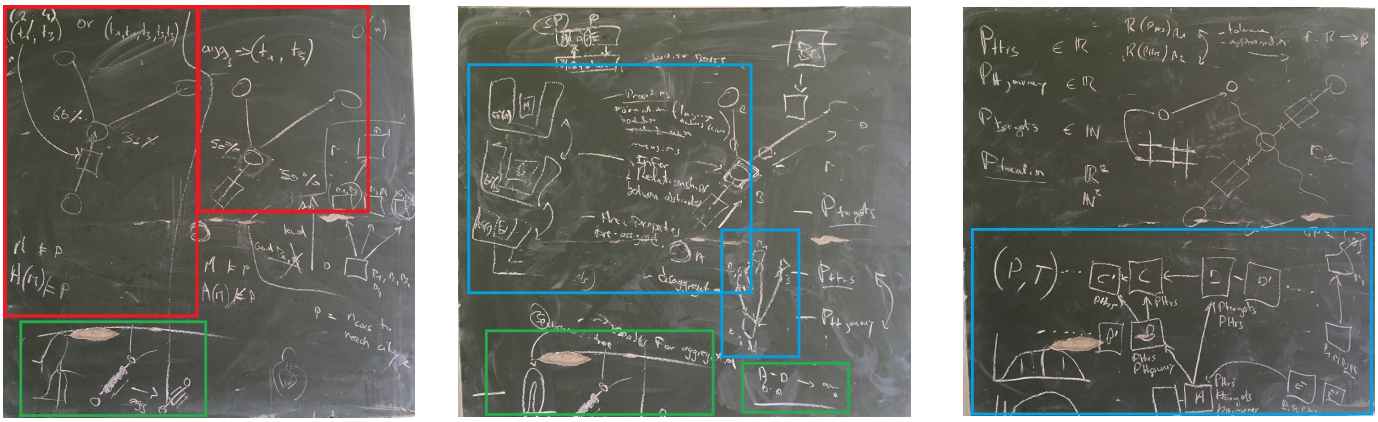


Fig. 2: Formalization of the sketching process.



(a) First iteration: examples and semantics. (b) Second iteration: first decision proposal. (c) Third iteration: decision reached.

Fig. 3: The sketches realized by three stakeholders at the CAMPaM 2019 workshop.

ideation phase and, in particular, the sketching activity, for the collaborative design and development of CPS.

A. Vision

We have argued that the ideation phase of projects should be supported with (digital) sketching interfaces. While providing the users with absolute freedom, such interfaces should provide a way to reuse the users' sketches later on in the design process. Furthermore, the interface should allow the (partial) evaluation of early-stage designs composed of sketches whose syntax and semantics are uncertain and/or incomplete.

We see three areas of future research that need to be addressed: 1) languages for the ideation phase are not predefined but co-constructed while the sketches evolve, thus (flexible) language engineering support is required for this to be performed in a structured way; 2) workflows in the ideation phase are inherently flexible and such flexibility needs to be supported by languages and tools; and 3) an infrastructure is required to support the sketching activity in a meaningful way.

B. Model-Language Co-Design

While sketching, the languages used to (partially) specify the system are developed alongside the models that specify the system's structure and behavior. This *model-language co-design* is at the heart of our envisioned approach. To support this co-design, a way to incrementally build a language (both its syntax and semantics) is necessary.

For the syntax, when part of the sketch is identified as belonging to a certain language, that part should no longer be considered as "sketched", but as properly typed by a language (constructed on-the-fly). A possible technique to allow for this retyping of artefacts is a-posteriori typing [4], which can be used to change the type of the sketch (the model) as the modeling language evolves. All kinds of concrete syntax (including graphical and textual syntax) should be supported. Textual syntax is often used to sketch algorithms or to annotate graphical sketches and often evolves too from a free form to a more structured form. In any concrete syntax, partial parsers that evolve as sketches evolve have to be supported.

The semantics of the language will be inherently incomplete and contain "holes", which is usually not allowed in traditional language engineering, where tools such as compilers and interpreters assume that a full specification of the semantics is available. To perform this evolution in a structured way, the first step would be to allow the presence of attribute values with a certain degree of *uncertainty*. For instance, one way to consider this uncertainty could be that, instead of having precise values, the semantics of the language allow for *ranges* of possible values. Additionally, modelers should be able to specify that they do not yet know the values of certain attributes, leaving them empty. Such "models with holes" could be resolved if the system behind it allows for (semi-)automated design-space exploration [9], proposing possible

resolutions for the unspecified values that are consistent with the rest of the (partial) design.

In parallel to the co-construction of the *design language*, a *property specification language* is needed as well, which can (and should) evolve together with the design language, offering abstractions to describe properties related to the structure of the system and its (temporal) behavior. The ProMoBox approach [10] is used in traditional language engineering to construct a property language whose syntax is very close to the syntax of the design language. When sketching, a design language is co-designed with the example models; this means that if we sketch properties, the property language is similarly co-designed. The relation between the sketched design language and sketched property language requires further research; furthermore, allowing for "models with holes" should also allow for property specifications over such models. Conversely, "properties with holes" (where there is uncertainty in the properties) also need to be allowed.

Ultimately, *sketching* and *modeling* are very similar in nature; engineers want to build abstractions of the system and evaluate them according to a set of properties. There is an increasing level of accuracy and fidelity once the design evolves from a sketch to a full-fledged model, but the activities we can perform with them should be the same. In that sense, sketching tools should support *full system analysis* from very early on, to already give the engineers an idea about the viability of their early design candidates.

C. Flexible Workflows

Current modeling tools not only have strict syntax rules; they also expect the users to follow a certain workflow (often implicitly defined in documentation). In sketching interfaces, such workflows cannot be rigid and should give users as much freedom as possible for exploring alternatives. However, to support the evaluation of sketches, underlying processes are required to make assumptions of the produced artefacts. While the sketches evolve, engineers might identify parts that they recognize such as use case-like diagrams, partial dataflows, or mechanical drawings. When these sketches are progressively typed and their semantics are partially described during the sketching process, the partial and/or incomplete model that the sketches represent could be evaluated.

Currently, our workflow model in Figure 2 does not support languages that evolve over time, where they are types of other artefacts in the workflow. We need this support to describe the relation between the example models (sketches) and their language, that evolve in tandem. To properly enact this flexible workflow a way of describing this evolution needs to be found.

D. Infrastructure

An intuitive sketching interface is important. To mimic the behavior of a whiteboard, a smartboard could be used to sketch using a pen-like object (which is the most widely used in collaborative settings, such as a brainstorm meeting). Closely related are tablets, which can be used in small collaborative settings or when sketching individually. The most general platform, however, is the web browser, which can be run on almost all devices.

Regardless of the medium used for sketching, collaboration also has to be taken into account when developing the necessary infrastructure for sketching. In a physical meeting, where people are in the same place at the same time, a smartboard with the proper tooling installed on it can be used to perform the sketching, and conversations take place in real-time. When the meeting is done remotely, because the team is located across different locations, support has to be provided for a collaborative tool: either this tool provides one "master screen" that is shared among the participating groups, or it allows for real-time collaborative editing, a difficult problem to solve in model-driven engineering [5]. Last, if the collaboration occurs at different points in time, support for version control has to be provided to store the history of the sketches in a central repository. This requires support for merging sketches.

VI. Related Work

A number of visual sketching tools that allow designers to build example models without restrictions have been developed. Most of these tools are research prototypes and implement an example-based approach towards language development: instead of building the language first, and then creating models in that language, the tool allows users to sketch their models in a free-hand editor, and annotate them in a way that allows the tool to derive a metamodel.

FlexiSketch [19] supports large-screen touch devices and smartboards. Users can sketch shapes and annotate them with a type. The (implicit) metamodel of the language is constructed incrementally while the user introduces new sketched elements and types. At some point, the user can choose to lock the metamodel, after which the interface becomes a modeling tool, offering the vocabulary that was constructed in the sketching phase. A metamodel is stored for each sketch; metamodels can be merged.

MLCBD [3] infers a metamodel algorithmically from a set of example sketches. It allows for users to register shapes that need to be recognized; sketched shapes are then classified when they are recognized. Once the (implicit) metamodel has been generated, a number of example models are automatically generated and the user is asked to validate them. Once the metamodel is validated by the user, it can be used to generate a domain-specific editing interface.

Scribbler [1] provides a free-hand sketching interface. Users can register shapes to be recognized, and a training interface is provided to teach the tools how to recognize a specific shape. Scribbler requires the user to define the metamodel of the language manually and map the recognized sketches to it.

metaBup [7] generates an explicit metamodel automatically from a set of sketches. It relies on an external drawing tool, in which users can create their sketches; it is technology-agnostic and provides a neutral sketching metamodel, to which the imported sketches are transformed. A mapping of sketched elements to types needs to be manually provided.

Model Workbench [12] is a sketching interface used (mainly) for textual languages: after every key stroke, the structure of the entered text is analyzed. Tokens and literals can be identified, but to make the distinction between the different types of tokens (keywords, identifiers, references), manual user input is required. Similarly to metaBup, the generated metamodel can be accessed and changed by the user.

The discussed tools only focus on syntax: no sketching of the semantics of the language is supported. Moreover, they do not allow for multiple concrete syntaxes to be sketched. Most tools make an explicit distinction between sketches (example models) and instance models. There is no integrated sketching/modeling environment where the language gradually evolves from an imprecise, sketched language with underspecified constraints and semantics to a fully specified language that can be used for detailed system design. We aim to bridge that gap by viewing the modeling activity not as distinct from the sketching activity, but a natural evolution of it. At any point in time, a modeling environment might allow for the user to start sketching parts of the system that are then later refined.

VII. Conclusion

This paper presents a vision to support the ideation phase, and specifically the sketching activity, for designing complex (cyber-physical) systems using a multi-paradigm modeling approach. We start from the observation that sketching is already performed in several domains, such as mechanical design. These sketches are always informal and are often not used in later design cycles. However, observing such sketches proved insightful and shows that the vocabulary used by the engineers in their sketches resembles a (partial) language for drawing examples to reach a common understanding and agreement; semantics of the language are sketched to convey its meaning. Conversely, traditional modeling and simulation tools require rigidly defined models before being able to simulate and analyze them. We propose to evolve current language engineering techniques to support sketching on three levels: model-language co-design, flexible workflows, and infrastructure. Our approach differs from current state-of-the-art sketching tools for model-driven engineering, as they typically take into account only syntax, and/or focus on the graphical aspect of the sketches to implement sketch recognition.

In future work, we plan to study the sketching process in more detail, to validate and potentially adapt or extend our formalization. We plan to additionally study the role of sketching in the requirements management, analysis, and design phases of project, especially in relation to existing, well-defined development processes. Once the languages, processes, and relation to other activities are validated, we plan an initial implementation of a sketching tool based on the recommendations presented in this paper. Its usefulness can then be evaluated with a number of user studies in different domains, with different user groups.

Acknowledgments

We thank the members of group number 5 (Romain Franceschini, Antonio Cicchetti, Moharram Challenger, and Joachim Denil) of the CAMPaM 2019 workshop, working on adaptive abstraction, for allowing us to use their sketches and for their time explaining them to us at the workshop.

References

[1] Christian Bartelt, Martin Vogel, and Tim Warnecke. Scribbler: From collaborative sketching to formal domain specific models and back again. In *Joint Proceedings of MODELS'13 Invited Talks, Demonstration Session, Poster Session, and ACM Student Research Competition*, volume 1115, 10 2013.

[2] M. Book and A. van der Hoek. Sketching with a purpose: Moving from supporting modeling to supporting software engineering activities. In *Proc. of the 11th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, pages 93–96, 2018.

[3] Hyun Cho. *A Demonstration-based Approach for Domain-specific Modeling Language Creation*. PhD thesis, University of Alabama, Tuscaloosa, AL, USA, 2013. AAI3562407.

[4] Juan de Lara and Esther Guerra. A posteriori typing for model-driven engineering: Concepts, analysis, and applications. *ACM Trans. Softw. Eng. Methodol.*, 25(4):31:1–31:60, May 2017.

[5] Mirco Franzago, Davide Di Ruscio, Ivano Malavolta, and Henry Muccini. Collaborative model-driven software engineering: a classification framework and a research map. *IEEE Transactions on Software Engineering*, 44(12):1146–1175, December 2018.

[6] Ben Jonson. Design ideation: the conceptual sketch in the digital age. *Design Studies*, 26:613–624, 2005.

[7] Jesús J. López-Fernández, Jesús Sánchez Cuadrado, Esther Guerra, and Juan Lara. Example-driven meta-model development. *Softw. Syst. Model.*, 14(4):1323–1347, October 2015.

[8] Levi Lúcio, Sadaf Mustafiz, Joachim Denil, Hans Vangheluwe, and Maris Jukss. FTG+PM: An integrated framework for investigating model transformation chains. In *SDL 2013: Model-Driven Dependability Engineering*, pages 182–202. Springer Berlin Heidelberg, 2013.

[9] Bart Meyers, Joachim Denil, Ken Vanherpen, and Hans Vangheluwe. Enabling design-space exploration for domain-specific modelling. In *Proceedings of the Model-driven Approaches for Simulation Engineering Symposium*, Mod4Sim '18, pages 5:1–5:13, San Diego, CA, USA, 2018. Society for Computer Simulation International.

[10] Bart Meyers, Hans Vangheluwe, Joachim Denil, and Rick Salay. A framework for temporal verification support in domain-specific modelling. *IEEE Transactions on Software Engineering*, PP:1–1, 07 2018.

[11] P. J. Mosterman and Hans Vangheluwe. Computer automated multiparadigm modeling: An introduction. *Simulation*, 80(9):433–450, September 2004.

[12] Bastian Roth, Matthias Jahn, and Stefan Jablonski. Rapid design of meta models. *International Journal on Advances in Software*, 7, January 2014.

[13] Nayan Ruparelia. Software development lifecycle models. *ACM SIGSOFT Software Engineering Notes*, 35:8–13, May 2010.

[14] Bernhard Schätz. The role of models in engineering of cyber-physical systems challenges and possibilities. In *CPS20: CPS 20 years from now - preproceedings*, pages 16–21, April 2014.

[15] Martina Schütze, Pierre Sachse, and Anne Römer. Support value of sketching in the design process. *Research in Engineering Design*, 14(2):89–97, 2003.

[16] Jami J. Shah, Noe Vargas-Hernandez, Joshua D. Summers, and Santosh Kulkarni. Collaborative sketching (C-Sketch) — an idea generation technique for engineering design. *The Journal of Creative Behavior*, 35(3):168–198, September 2001.

[17] Pieter Jan Stappers and James M. Hennessey. Toward electronic napkins and beermats: Computer support for visual ideation skills. In Ray Paton and Irene Neilson, editors, *Visual Representations and Interpretations*, pages 220–225, London, 1999. Springer London.

[18] IM Verstijnen, C van Leeuwen, G Goldschmidt, R Hamel, and JM Hennessey. Sketching and creative discovery. *Design Studies*, 19(4):519 – 546, 1998.

[19] Dustin Wüest, Norbert Seyff, and Martin Glinz. FlexiSketch: a lightweight sketching and metamodeling approach for end-users. *Software & Systems Modeling*, 18(2):1513–1541, Apr 2019.